REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	14 March, 1995	Final: 5/20/94 - 10/19/94

4. TITLE AND SUBTITLE
A Study of Computational Requirements for Problems in Pattern Recognition

5. FUNDING NUMBERS
G: N0001494-1-0804

6. AUTHOR(S)
Nelson Morgan, Jerome Feldman

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
International Computer Science Institute
1947 Center St., Suite 600
Berkeley, CA 94704-1198

8. PERFORMING ORGANIZATION REPORT NUMBER
1-25-01

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Dept. of the Navy
Office of Naval Research
Seattle Regional Office
1107 NE 45th St., Ste. 350
Seattle, WA 98105-4631

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
4330/11 ONR 247

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

During the contract period we:

1. Considered connectionist computation, particularly asit has been used for applications in speech recognition, and developed perspectives on machine requirements for this task.

2. Analyzed the computational requirements for image understanding tasks, particularly in the context of the developing Arpa Image Understanding Environment.

3. Studied the relevance of object-oriented simulation software for these tasks.

14. SUBJECT TERMS	15. NUMBER OF PAGES
connectionist computation object-oriented simulation software speech recognition	7
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
			SAR

Final Report: A Study of Computational Requirements for Problems in Pattern Recognition

Nelson Morgan and Jerome Feldman
International Computer Science Institute
1947 Center Street
Berkeley CA 94704

March 14, 1995

1 Introduction

This report summarizes the work performed under ARPA/ONR grant N00014-94-1-0804. During the contract period we:

1. Considered connectionist computation, particularly as it has been used for applications in speech recognition, and developed perspectives on machine requirements for this task.
2. Analyzed the computational requirements for image understanding tasks, particularly in the context of the developing Arpa Image Understanding Environment.
3. Studied the relevance of object-oriented simulation software for these tasks.

This is the summary report of the findings of these studies.

2 Project Summary

We have studied a number of pattern recognition applications to determine what features are required for a scalable accelerator system that is specialized for such applications. We believe that it is possible to develop a family of hardware and software tools that can range from single chip accelerators to stand-alone specialized super-computers for these tasks. Our past focus has primarily been on connectionist algorithms, but we also have included a number of algorithms that are not traditionally expressed in this way, such as dynamic programming.

System level hardware and software support requirements have been examined for a number of aspects of spoken language processing and vision. The potential for these capabilities already exists in the CNS design that we have been working on, but we felt that it would be helpful to further examine such requirements, both for our own design and for the future designs of other machines.

Specialized hardware has suffered from its inability to do all the computations needed for an application. This often leads to poor performance on the complete task despite very good speed on some subtasks. We have studied the utility of incorporating a RISC core on the signal processing chip to increase the range of tasks that can be accelerated. We have investigated the need for non-multiply-accumulate vector routines for vectorized nonlinear functions, and have in particular considered the application tos image understanding. To support users with a wider range of applications, the need for object-oriented simulation software has also been further examined.

3 Applications

It is difficult to design a parallel machine to be equally effective for all applications. Ideally the design team would carefully examine proposed tasks and determine what features were required to optimize performance (within cost constraints) for the target applications. Unfortunately, for many cases there is a "chicken and egg" problem; that is, the optimum computer cannot be designed until we have experience with the application, but we cannot gain this experience until we have a machine that can run the application efficiently.

A feasible solution to this dilemma is to observe the requirements for smaller tasks that we believe to be similar to our end goals, as well as to analyze those cases which have no good working precedent. A machine is then designed to perform well on both sets of these problems. Iterations of this process should result in better solutions both at the algorithmic and architectural levels.

We have considered a number of application targets for this study. These range from obvious extensions of tasks we have run on parallel computers at ICSI through bona fide applications that have never been implemented on fast hardware. In addition, we have looked at abstract target applications that we believe to be representative of classes of problems which have withstood conventional analysis.

3.1 Connectionist Processing

We emphasized two sample subproblems which were required for a number of connectionist applications, but which in particular were necessary for our use of networks for phonetic probability estimation:

1. A general mechanism for fast implementation of nonlinear functions; in particular, the log of the sum of two exponentiated values. This function is used in a number of ways, including the Viterbi (dynamic programming with log probabilities) and the softmax function that is often used at the output of the network probability estimator.

2. Fast approaches to workstation implementation of the neural network recall phase. While training is still sufficiently computationally intensive to require specialized hardware for the size of multilayer perceptrons we are using in speech recognition, it should be possible to streamline the recognition code so that it can be done on a modern workstation without the aid of accelerators. Furthermore, such code would be a more honest benchmark for comparison with specialized hardware, since the latter is almost always made acceptably efficient through the use of very specialized software.

3.1.1 Nonlinear function approximation

The family of designs that we have been developing in the CNS-1 project, based on the Torrent architecture, can support almost any function taking a single 16-bit fixed-point input and returning a 16-bit fixed-point output. Inputs and outputs must each be over a fixed range, meaning that the binary points of the input and outputs must be in particular locations for a given function implementation. (Pseudo-floating-point is not feasible.) Functions are represented as a sequence of line segments (that is, piece-wise linear). An entire function is encoded in a table of 8192 bytes. For "well-behaved" functions, the error is usually better than 1 ulp (unit in the last place) of the function result - typically the error is around 0.8 ulps. This error increases for functions whose slope exceeds about +/-64 (relative to the input and output scale), or whose second derivative exceeds about +/-128 (again, relative to the input and output scale).

For example, using the table representation, one can easily implement an arctangent function having fixed-point inputs over the range -32 to +32 and outputs between -pi/2 and +pi/2. A square root function, on the other hand, is inherently problematic because the slope and curvature are both infinite at 0. (A table can be used to code most of the square root function, with special case code taking care of small inputs, however.)

Construction of a function table is made trivial by two library routines: The first takes a double-precision floating-point function as input and returns a table giving a piece-wise linear representation of an equivalent 16-bit fixed-point function. The second library routine takes both of the above as inputs and calculates the worst-case error exhibited by the approximation. Another set of library routines perform function evaluations through lookups in the table. Both scalar and vector lookups are provided. Using the library lookup routine, vector fixed-point function evaluation requires only about 1 cycle per element for long vectors.

Taking advantage of this environment, we implemented fast scalar and vector fixed-point library routines for calculating exp(x) and log(exp(x)+exp(y)). The table-based functions described above do not implement these library functions directly: obviously, log(exp(x)+exp(y)) requires more than one input, and both functions are evaluated to more than 16 bits, in fact. Table-based 16-bit functions are used internally, however. Consquently, the vector library routine for each of these functions takes about 2 cycles per element for long vectors.

3.1.2 Fast C matrix-vector libraries for workstations

Most of our neural network application code is written using either ANSI C or C++, and a significant number of the most time consuming portions of the code can be described in terms of well known linear algebra operations such as matrix-matrix multiply.

Generating efficient code for commercial cached RISC architectures from

naive matrix-vector operations coded in C is difficult, and while some recent research compilers have made significant progress, commercial C/C++ compilers often give disappointing results. The C language presents considerable barriers to memory aliasing analysis through its use of unrestricted pointers as array indices. This limits the extent to which even sophisticated C compilers can improve performance without explicit user annotation (pragmas).

An alternative approach to achieving portable high performance is through the use of standard matrix-vector libraries. If a portable interface is defined, specific library implementations can optimize these basic operations for a given architecture, and can use hand assembly coding or other high performance languages. A further major benefit from the use of standard library routines is that code development and maintenance is simplified, since now large pieces of code can be reused from a library. We are interested in the use of application specific hardware to accelerate certain applications and the use of standard libraries can significantly simplify the task of porting user code to new hardware.

In the numerical analysis community, which is dominated by use of the FORTRAN language, a set of basic linear algebra subroutines (BLAS) has been defined that captures the most important primitive operations in linear algebra. System vendors often distribute carefully hand crafted assembler code to implement these routines, but for certain machines such libraries may be nonexistent or expensive. Portable BLAS sources are available in FORTRAN, but these cannot be used on systems without optimizing FORTRAN compilers or without good code development tools for managing mixed C and FORTRAN programs. Finally, while the BLAS library provides extensive functionality in the fields of linear algebra, it lacks important functions required for our connectionist research, such as various non-linear activation squashing functions.

We have begun work on a set of portable, high-performance, ANSI C libraries (PHIPAC). The intent is to provide portable ANSI C source code that will yield excellent performance on various matrix-vector operations for machines containing pipelined and/or superscalar RISC processors with cached memory hierarchies. The method we use is to hand write C code that explicitly performs optimizations that unsophisticated compilers will not perform, and to also perform optimizations that even sophisticated compilers cannot perform without explicit code annotations. The most important optimizations we make include global register allocation, loop unrolling, and various loop transformations for improved cache performance. We concentrate on providing higher level restructuring, while relying on the machine's native C compiler to perform detailed instruction selection, register allocation, and instruction scheduling.

Initial results are encouraging. A portable C matrix-matrix multiply routine has been written and tested on three separate workstation architectures. On a Sparcstation-20/61 we achieved performance of around 42 MFLOPS with single-precision matrices of rank 256, this represents 70% of peak performance for this workstation. For comparison, a naive triply-nested matrix-matrix multiply routine achieves roughly 9 MFLOPS on the same workstation. For an IBM

RS/6000 model 590 workstation, we achieve performance of over 240 MFLOPS, close to the 245 MFLOPS achieved by the IBM-supplied machine-specific ESSL libraries, and roughly 90% of machine peak performance. On an older SGI Indigo R4000 running at 100 MHz we achieved 25 MFLOPS representing 37% of peak performance. Note that the combination of a smaller number of floating point registers, and non-pipelined and long functional unit latencies makes it difficult to saturate the floating point unit of the R4000.

We intend to continue development of this library and to make it publically available.

3.2 Image Understanding

Accelerators have a long history in image understanding, but are almost always limited to the lowest levels of the problem. The availablity of tightly coupled systems, like our CNS design, permits accelration of the full range of computationally intensive image understanding tasks. We studied the requirements of an automated vehicle task in detail and showed that all of the hard parts of the computation can be speeded up significantly on the CNS architecture.

A second major investigation concerned computational geometry. Modern image understanding systems use very sophisticated computational geometry routines and these are quite different in kind from the image processing of early vision or the statistical techniques used in other vision tasks. We established a working relationship with the leaders of the Darpa effort on computational geometry for the Image Understanding Environment project and started to work out their key computational requirements. At the end of this contract, these requirements were still not fully specified, but it seemed clear that the CNS architecture could accelerate the known requirements and similar tasks. Accelerators for the full Image Understanding Environment will require a wide range of capabilities and should be developed in close collaboration with the application groups.

3.3 Object-oriented Software Support

It has been much easier to build accelerator hardware than to progam it effectively. Our group has always developed software support in parallel with the hardware and this has been one of the keys to our success. As parallel hardware and accelerators are applied to ever more complex tasks, the role of software support becomes even more important. As part of our study we examined the software needs of advanced systems in speech and image understanding. We concluded that the requirements are too complex to be met by fixed software packages or special purpose simulators. However, the development of modern object-oriented languages allows for the development of packages that can be efficent in execution, easy to use and fairly easy to recongigure for additonal tasks. Part of our work included detailed studies of mapping connectionist and

other computations to parallel hardware. We also studied the role of parallel object-oriented languages in building flexible and efficient systems.

4 Conclusions

This was a relatively short study, but a number of things were quite clear to us at its conclusion, including:

1. An architecture with an embedded RISC engine in addition to a fixed-point vector coprocessor is potentially useful for a range of tasks beyond the neural network training for speech that was our original principal task. In particular, general capabilities for vector nonlinear function evaluation and specific capabilities for image understanding tasks could be accelerated with such an approach.

2. All such developments should be fairly compared with modern workstation performance. In particular, accelerator hardware throughput is generally specified for extremely tuned code, while general purpose machines are usually characterized in terms of performance with non-optimized code. We recently tested a commercial accelerator for neural networks on one of our speech training problems and achieved only 4% of its rated peak performance. A similar task was carefully coded for a workstation and ran at 70% of its peak performance. It may well be true that extensive recoding could bring back much of this lost performance, but such development on specialized hardware can be time-consuming, and the experiments we report above showed that this attention to code efficiency can also have significant effects on workstation performance on target tasks. All of this is not intended as an argument against specialized accelerators, but only to caution that acceleration claims should be tested by comparison with workstation code that has been similarly optimized.

3. The programming of accelerator hardware can be simplified by the use of object-oriented techniques, though particular simulators or fixed software packages are necessarily insufficient.

We now are proceeding to work on the development of further software libraries to help in a host of applications such as the ones considered above. Our own chip design has gone out for fabrication, so that we hope to soon be doing research on this topic using the working silicon rather than the simulation.